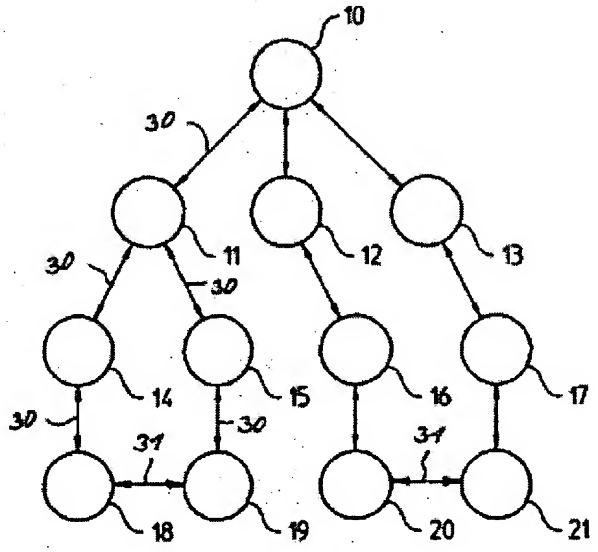
Verfahren zum synchronisierten Hochlauf einer Steuerung

Publication number: DE19924461
Publication date: 2000-11-30
Inventor: RUTKOWSKI CHRISTIAN (DE); KLUGHAMMER THOMAS (DE)
Applicant: HEIDENHAIN GMBH DR JOHANNES (DE)
Classification:
- international: ***G05B19/418;*** **G05B19/418;** (IPC1-7): G05B19/414
- european: G05B19/418N
Application number: DE19991024461 19990528
Priority number(s): DE19991024461 19990528

Also published as:
WO0073864 (A1)
EP1185910 (A1)
US6988191 (B1)
EP1185910 (A0)
EP1185910 (B1)

Abstract of **DE19924461**

The aim of the invention is to provide a numerical control of a machine tool that is easy to maintenance and update while being controlled by increasingly complex programs. To this end, the programs for numerical controls have increasingly object-oriented structures. According to said structure, applications (10), processes (11 to 13), threads (14 to 17) and modules (18 to 21) are represented by respective objects (10 to 21). When the numerical control is run up, those objects (10 to 17) are produced and in turn produce further objects (11 to 21), every object (10 to 21) assuming a number of defined states. Specifically, every object (10 to 17) initiates in all objects (11 to 21) produced by it the initialization function realized for said state by the respective object (10 to 21) and assumes the next defined state only if the object itself and all objects (10 to 21) produced by it have completed all functions. This state is reported to the producer object (10 to 17) until the feedback reaches the application object (10). The application object (10) itself assumes said state and initiates the next step of the run-up or terminates the run-up once the last step of the run-up has been completed.



(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 199 24 461 A 1**

(51) Int. Cl.[7]: G 05 B 19/414

(21) Aktenzeichen: 199 24 461.8
(22) Anmeldetag: 28. 5. 1999
(43) Offenlegungstag: 30. 11. 2000

DE 199 24 461 A 1

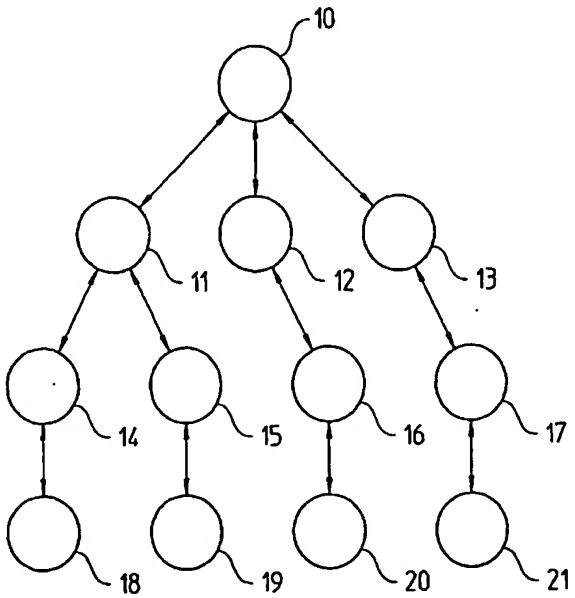
(71) Anmelder:
Dr. Johannes Heidenhain GmbH, 83301 Traunreut, DE

(72) Erfinder:
Rutkowski, Christian, Dipl.-Phys., 83278 Traunstein, DE; Klughammer, Thomas, Dipl.-Inform.(FH), 83339 Chieming, DE

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Verfahren zum synchronisierten Hochlauf einer Steuerung

(57) Um bei einer zunehmenden Komplexität der Programme für numerische Steuerungen für Werkzeugmaschinen eine einfache Wartbarkeit und Erweiterbarkeit sicherzustellen, weisen die Programme für numerische Steuerungen zunehmend eine objektorientierte Struktur auf. Dabei werden Applikationen, Prozesse, Threads und Module jeweils durch ein Objekt repräsentiert. Bei einem Hochlauf werden zunächst diejenigen Objekte erzeugt, die weitere Objekte erzeugen, und es nimmt jedes Objekt eine Anzahl definierter Zustände an. Insbesondere startet jedes Objekt bei allen von ihm erzeugten Objekten die für diesen Zustand durch das jeweilige Objekt realisierte Initialisierungsfunktion und nimmt den nächsten definierten Zustand dann an, wenn es selbst und alle von ihm erzeugten Objekte sämtliche Funktionen vollständig ausgeführt haben. Dieser Zustand wird an das erzeugende Objekt gemeldet, bis diese Rückmeldung beim Applikationsobjekt eintrifft. Das Applikationsobjekt nimmt dann selbst diesen Zustand an und löst den nächsten Schritt des Hochlaufs aus bzw. beendet den Hochlauf, nachdem der letzte Schritt des Hochlaufs ausgeführt wurde.



DE 199 24 461 A 1

Beschreibung

Die Erfindung betrifft ein Verfahren zum synchronisierten Hochlauf einer Steuerung nach dem Oberbegriff des Anspruchs 1.

Die EP 0 524 344 B1 offenbart eine konfigurierbare Werkzeugmaschinensteuerung, die aus mehreren aufgabenorientierten Einheiten besteht, beispielsweise einem numerischen und einem speicherprogrammierbaren Steuerteil, einer Bedienereinheit und einem Kommunikationsbereich mit Netzwerkschnittstelle. Weiterhin ist mindestens ein in Soft- oder in Soft- und Hardware realisiertes Funktionsobjekt, das eine Funktion durchführen kann vorgesehen. Dieses Funktionsobjekt untergliedert sich in einen Prozedurteil, einen Kommunikationsteil und evtl. einen Bedienteil. Zusätzlich ist mindestens ein Objektmanager vorhanden, der mindestens zwei Funktionsobjekte verwaltet, insbesondere deren Nachrichtenaustausch synchronisiert. Diese Steuerungsstruktur wird auf mindestens einer Datenverarbeitungsanlage realisiert, welche die Daten der Funktionsobjekte und Objektmanager bearbeitet und selbst als aufgabenorientierte Einheit ausgebildet ist.

Aus der EP 0 524 344 B1 ist kein systematisches Verfahren zum Starten einer objektorientiert realisierten Steuerung bekannt.

Aus der EP 0 657 043 B1 ist eine objektorientierte Steuerung für Werkzeugmaschinen bekannt, bei der aus Objektklassen eine Reihe von Objekten gebildet werden. Ausgehend von der in der EP 0 524 344 B1 offenbarten Struktur einer Steuerung werden in der EP 0 657 043 B1 die konkret erforderlichen Objektklassen und Objekte offenbart, die erforderlich sind, um eine herkömmliche Funktionalität einer Steuerung zu realisieren. Dazu gehören beispielsweise Objektklassen für Bearbeitungs-, Geometrie, Kinematik- und Technologiedatentypen ebenso wie Steuerdatentypen und eine Objektklasse Ablaufsteuerung. Von jeder Objektklasse kann eine beliebige Anzahl Objekte gebildet werden, die jeweils einen eigenen Datenbereich, einen Botschaftenmechanismus zur Kommunikation mit anderen Objekten und einen Prozedurteil zur Ausführung von Methoden zur Bearbeitung, Kinematik, Geometrie oder Technologie beinhalten. Durch die Ablaufsteuerung werden die Benutzereingaben interpretiert und führen zur Aktivierung der ausgewählten Objekte. Die ausgewählten Objekte kommunizieren untereinander und bilden durch diese netzwerkartige Verknüpfung eine ablauffähige Funktionseinheit der Steuerung.

Auch aus der EP 0 657 043 B1 ist kein systematisches Verfahren zum Starten einer objektorientiert realisierten Steuerung bekannt.

Aus der EP 0 717 866 B1 ist ein CNC-Steuerungssystem bekannt, das ein objektorientiertes Programm beinhaltet, in welchem Objekte objektorientierte Nachrichten austauschen. Im objektorientierten Programm sind die Objekte in Klassen unterteilt, beispielsweise eine Prozessklasse, die Objekte für Bearbeitungsprozesse wie Bohren, Gewindeschneiden, Räumen usw. beinhaltet, die durch Maschinenkomponenten ausgeführt werden. Dabei beinhaltet eine Klasse immer ähnliche Objekte, das heisst Objekte, die in ihrer Grundstruktur übereinstimmen. Aufgrund der einheitlichen Grundstruktur der Objekte einer Klasse besteht die Möglichkeit, dass beim Erstellen neuer Objekte ausgewählte Eigenschaften der jeweiligen Klasse an das neue Objekt vererbt werden. Beispielsweise weist ein Bohr-Objekt eine Tiefe und einen Durchmesser auf, die von einem anderen Objekt der Prozessklasse geerbt werden können, beispielsweise einem Gewindeschneid-Objekt, das ebenfalls eine Tiefe und einen Durchmesser aufweist. Eine weitere Objektklasse weist Maschinenkomponenten, wie beispielsweise eine Spindel, Achsen, einen Drehtisch usw., auf. Weiterhin sind Objektklassen für das Kernel mit einem Motion und einem Logic Controller als Objekte, für Plattform Services, das Operating System und die Device Driver vorgesehen. Im Betrieb der Steuerung ist es erforderlich, dass Nachrichten zwischen den einzelnen Objekten ausgetauscht werden. Für einen Bohrvorgang wird beispielsweise vom Bohr-Objekt eine Nachricht die Drehzahl des Bohrers betreffend an das Objekt Spindel übertragen, weiterhin werden Nachrichten die Position des Loches betreffend an die Objekte der beteiligten Achsen übertragen usw.. Dabei ist ein Standard-Interface für die Nachrichten vorgesehen, damit diese einen universellen Aufbau aufweisen und unabhängig von den beteiligten Objekten ausgestaltet werden können. Diese Standard-Schnittstelle zum Nachrichtenaustausch zwischen Objekten wird bei einem Objekt, das Nachrichten bezüglich der Bewegung empfängt oder sendet durch einen Softwarekern realisiert, der in Echtzeit arbeiten soll und Nachrichten empfängt und sendet.

Ein Verfahren für einen synchronisierten Hochlauf einer objektorientierten Steuerung wird dabei nicht offenbart.

Aufgabe der vorliegenden Erfindung ist es daher, ein Verfahren für einen Start einer Applikation einer numerischen Steuerung anzugeben, bei dem synchronisiert deren Prozesse, Threads und Module in eine Struktur eingebunden und initialisiert werden und bei dem anschliessend in den Normalbetrieb übergegangen wird. Es soll sichergestellt werden, dass beim Übergang in den Normalbetrieb insbesondere die Kommunikationskanäle vorhanden sind, über die Module miteinander kommunizieren, die zu unterschiedlichen Threads oder Prozessen gehören.

Diese Aufgabe wird durch ein Verfahren mit den im Anspruch 1 angegebenen Merkmalen gelöst.

Weiterbildungen und vorteilhafte Ausgestaltungen des erfindungsgemässen Verfahrens sind den abhängigen Ansprüchen zu entnehmen.

Das erfindungsgemässe Verfahren weist den Vorteil auf, dass die Software einer Steuerung für eine Werkzeugmaschine in Form einer Applikation mit den zugehörigen Prozessen, Threads und Modulen bis zur Betriebsbereitschaft der Steuerung synchronisiert gestartet wird. Dabei wird in einem ersten Initialisierungsschritt die Applikation in Form einer oder mehrerer ausführbarer Dateien in den Arbeitsspeicher geladen und es werden die einzelnen Objekte, wie beispielsweise Prozesse, Threads und Module erzeugt. Sobald alle Objekte erzeugt wurden, werden in einem zweiten Initialisierungsschritt die Objekte initialisiert, beispielsweise mit Parameterwerten. Sobald dieser Initialisierungsschritt abgeschlossen ist, können weitere Initialisierungsschritte folgen, bis in einem letzten Initialisierungsschritt die gesamte Applikation für den Betrieb freigegeben wird. Somit sind bei einem fehlerfreien Hochlauf alle Applikationen, Prozesse, Threads und Module erzeugt, initialisiert und die Steuerung ist betriebsbereit. Sollten Fehler in einem Objekt auftreten, ist nach einem derartigen synchronisierten Hochlauf bekannt, welche Prozesse, Threads oder Module fehlerhaft sind. Ein weiterer Vorteil besteht darin, dass dem Anwender gemeldet wird, falls dies für eine benötigte Applikation, einen Prozess, einen Thread oder ein Modul nicht der Fall sein sollte.

Die Erfindung wird im folgenden anhand der in der Zeichnung dargestellten Ausführungsformen näher erläutert. Die Figur zeigt:

Fig. 1 eine mögliche schaltungstechnische Realisierungsform einer erfindungsgemässen Steuerung,

Fig. 2 eine mögliche Struktur einer objektorientierten Applikation. Die Erfindung soll im folgenden anhand einer einfachen numerischen Steuerung für eine Werkzeugma-

schine erläutert werden.

Die Steuerungsanordnung weist mindestens einen Massenspeicher **1** auf, der über einen bidirektionalen Bus **2** mit einem Prozessor **3** verbunden ist. Dadurch kann der Prozessor **2** schreibend und lesend auf den Massenspeicher **1** zugreifen. Der Massenspeicher **1** kann durch einen magnetischen oder optischen Speicher oder eine Kombination mehrerer unterschiedlicher Speicher realisiert werden. Weiterhin kann der Massenspeicher **1** auswechselbare Speichermedien, wie beispielsweise Compakt Disks, aufweisen. Über den Bus **2** besteht auch eine bidirektionale Verbindung mit einem Arbeitsspeicher **4**, der als Schreib-/Lesespeicher ausgestaltet ist und eine kürzere Zugriffszeit als der Massenspeicher **1** aufweist. Der Arbeitsspeicher **4** kann – zumindest teilweise – auch in den Prozessor **3** integriert ausgeführt sein. Über den Bus **2** besteht weiterhin eine bidirektionale Verbindung zu einem Festwertspeicher **5**, der im Unterschied zum Arbeitsspeicher **4** nur eine kleine Speicherkapazität aufweist und als Lese-Speicher ausgestaltet ist. An den Bus ist ausserdem eine Reglerbaugruppe **6** angeschlossen, mit der wiederum die Antriebsmotoren für die Achsen und die Spindel verbunden sind.

Damit der Prozessor **2** der Steuerungsanordnung die verschiedenen Funktionen der Werkzeugmaschine steuern kann, muss die Software für die Werkzeugmaschine, die im folgenden mit Applikation bezeichnet und vom Prozessor **2** abgearbeitet wird, im Arbeitsspeicher **4** bereitgestellt werden. Da der Arbeitsspeicher **4** in der Regel kein permanenter Speicher ist, wird auf dem Massenspeicher **1** eine ausführbare Datei gespeichert, die ein Erzeugen der gesamten Applikation im Arbeitsspeicher **4** durch den Prozessor **2** ermöglicht, sobald die Steuerungsanordnung mit der Versorgungsspannung verbunden wird. Die Erzeugung der Applikation aus der im Massenspeicher **1** gespeicherten Datei wird im folgenden als Initialisierung bezeichnet, die meist in mehreren Schritten erfolgt.

Zu Beginn der Initialisierung wird die Struktur der Applikation erzeugt. Dabei werden die Daten der mindestens einen auf dem Massenspeicher **1** gespeicherten und zur Applikation gehörenden ausführbaren Datei in den Arbeitsspeicher **4** übertragen und abgearbeitet. Bei der Abarbeitung werden miteinander verknüpfte Objekte erzeugt. Die einzelnen Objekte können Kontext sein, wie beispielsweise Threads, Kommunikationskanäle, globale Speicher, oder es können Datenstrukturen sein, wie beispielsweise Objekte einer objektorientierten Struktur.

Eine Applikation wird in mehrere Prozesse untergliedert, wobei vorteilhaft jeder Prozess einen eigenen Adressraum im Arbeitsspeicher **4** besitzt. Es können sich aber auch mehrere oder alle Prozesse einen gemeinsamen Adressraum teilen. Jeder Prozess wird in Threads untergliedert und weist mindestens einen Thread auf. Der innerhalb eines Threads abgearbeitete Code ist in Module unterteilt, zu denen je eine definierte Datenstruktur gehört und in denen je mindestens eine konkrete Funktion der Steuerung realisiert ist, z. B. eine Tasche fräsen oder ein Gewinde schneiden usw. Die zur Applikation gehörenden Module, Threads und Prozesse sind derart miteinander verknüpft, dass sie auch über die Grenzen von Threads und Prozessen hinweg Daten austauschen können.

Um eine derartige Struktur der Applikation besonders einfach programmtechnisch zu realisieren, wird vorteilhaft eine objektorientierte Programmiersprache gewählt. Mittels der objektorientierten Programmiersprache wird die Applikation, jeder Prozess, jeder Thread und jedes Modul als ein Objekt ausgestaltet. Die Objekte können synchronisiert werden, weitere Objekt erzeugen und die erzeugten Objekte verwalten. Eine derartige Struktur und die genannten Funktionen können prinzipiell auch durch eine nicht objektorientierte Programmiersprache realisiert werden.

Wird die Steuerungsanordnung mit der Versorgungsspannung verbunden, wird zunächst deren Betriebssystem-Software in den Arbeitsspeicher **4** der Steuerungsanordnung geladen und anschliessend das Startprogramm der Applikation. Dieses Startprogramm wird anschliessend ausgeführt und steuert die synchronisierte Erzeugung zumindest der weiteren Prozesse. Alternativ kann durch das Startprogramm auch die Erzeugung der gesamten Struktur gesteuert werden.

Die vom Startprogramm benötigten Daten sind in einer separaten, einzulesenden Datei oder unmittelbar im Startprogramm selbst gespeichert. Falls diese Daten in einer separaten Datei gespeichert sind, lädt das Startprogramm auch diese Datei in den Arbeitsspeicher **4** und wertet sie aus. Die Daten bezeichnen mindestens eine auf dem Massenspeicher **1** gespeicherte, zu ladende ausführbare Datei, für einen Prozeß, der Applikation und optional Übergabeparameter der Prozesse, Informationen zur Synchronisation der Prozesse und Informationen über die bei der Initialisierung durchzuführenden Initialisierungs-Schritte. Diese derart definierten ausführbaren Dateien werden durch das Startprogramm in den Arbeitsspeicher **4** der Steuerung geladen und ausgewertet. Anschliessend können die Prozesse erzeugt werden.

Beim Abarbeiten des Startprogramms für den synchronisierten Hochlauf der Steuerung werden also nacheinander für jede Applikation die benötigten Dateien in den Arbeitsspeicher **4** geladen. Danach werden die Prozesse, die die Applikation bilden, erzeugt, anschliessend die Threads und letztendlich die zu den Modulen gehörenden Datenstrukturen. Dabei werden für die Erzeugung der zu jedem Prozess gehörenden Threads und der zu jedem Thread gehörenden Module weitere Daten benötigt. Die Daten zur Erzeugung der Threads können im Startprogramm, bzw. einer vom Startprogramm eingelesenen Datei, in einer ausführbaren Datei des Prozesses oder in einer jeweils eingelesenen Datei gespeichert sein.

Sinngemäß gilt das auch für die Module. Auch hier können die für die Erzeugung benötigten Daten bereits im Startprogramm, bzw. in einer vom Startprogramm geladenen Datei, oder in einer ausführbaren Datei oder einer eigenen Datei gespeichert sein. Der zum jeweiligen Modul gehörende Thread erzeugt aufgrund der Daten über ein Modul dieses Modul.

Dadurch entsteht eine baumartige Struktur der Objekte, an deren Spitze die Applikation auf der höchsten Hierarchie-Ebene steht. Die Applikation untergliedert sich in Prozesse, welche wiederum aus Threads bestehen. Auf der untersten Hierarchie-Ebene stehen die Module, die den Threads zugeordnet sind.

Die Verknüpfung zwischen den Modulen, Threads und Prozessen bis hin zur Applikation wird durch Kommunikationskanäle realisiert. Da die Module jeweils eine bestimmte Funktionalität aufweisen, werden für jedes Modul definierte Kommunikationskanäle benötigt, über die Daten von oder zu einem anderen Modul übertragen werden. Dies gilt entsprechend auch für die Kommunikationskanäle zwischen Threads und Prozessen.

In einer erfindungsgemässen Ausgestaltung weist jedes Objekt unter anderem einen Programmteil auf, der der Initialisierung dieses Objekts dient und für jedes Objekt individuelle Massnahmen zu dessen Initialisierung ausführt. Diese individuellen Massnahmen bestehen zumindest darin, dass das Objekt bei den ihm hierarchisch untergeordneten Objekten die Initialisierung veranlasst, auf die Ausführungsmeldungen aller untergeordneter Objekte wartet und dann an das übergeordnete Objekt, nach eigener korrekter

Initialisierung, eine Ausführungsmeldung weiterleitet. Zusätzlich können durch den Programmteil zur Initialisierung weitere Initialisierungsmassnahmen erfolgen.

Mittels einer objektorientierten Programmiersprache kann diese Zuordnung von Programmteilen besonders vorteilhaft realisiert werden.

Sobald ein Modul korrekt erzeugt wurde und sich als Modulobjekt in die objektorientierte Struktur eingefügt hat, sendet das Modulobjekt an das ihm hierarchisch übergeordnete Threadobjekt, das das Modulobjekt erzeugt hat, eine Ausführungsmeldung zurück. Daran erkennt das Threadobjekt, dass das Modulobjekt fehlerfrei erzeugt wurde. Sobald das Threadobjekt von jedem von ihm erzeugten Modulobjekt die Ausführungsmeldung erhalten hat und das Threadobjekt auch selbst korrekt erzeugt wurde, sendet es an das ihm übergeordnete Prozessobjekt ebenfalls eine Ausführungsmeldung. Sobald wiederum jedes Prozessobjekt von allen von ihm erzeugten Threadobjekten eine Ausführungsmeldung zur korrekten Erzeugung empfangen hat und das Prozessobjekt auch selbst korrekt erzeugt wurde, sendet das Prozessobjekt eine Ausführungsmeldung an das übergeordnete Applikationsobjekt.

Sobald das Applikationsobjekt von allen von ihm erzeugten Prozessobjekten aufgrund der empfangenen Ausführungsmeldungen eine korrekte Erzeugung erkannt hat, erfolgt der nächste Schritt der Initialisierung. Bis dahin müssen alle bereits erzeugten Objekte warten, damit keine Überschneidungen bei der Initialisierung auftreten, beispielsweise dass ein erstes Objekt mit einem zweiten Objekt kommunizieren will, obwohl das zweite Objekt noch nicht erzeugt wurde. Durch dieses Warten wird die Synchronisation der Initialisierung erreicht, da alle Objekte zu einem definierten Zeitpunkt einen definierten Zustand aufweisen. In einem typischen Multitasking Betriebssystem stehen zur Realisierung dieses Wartens bzw. dieser Synchronisation Mechanismen wie Semaphoren und Barrieren o. ä. zur Verfügung.

In einem zweiten Schritt der Initialisierung werden die Kommunikationskanäle zwischen den Objekten eingerichtet. Indem die Ausführung des im jeweiligen Prozessobjekt zur Initialisierung der Kommunikationskanäle vorgesehenen Programmteils veranlasst wird, werden die Kommunikationskanäle eines Prozessobjekts eingerichtet. Dies setzt sich entsprechend der bei der Erzeugung der Struktur bereits beschriebenen Vorgehensweise auch für Threads und Module fort, bis alle Module ihre Kommunikationskanäle erzeugt haben.

Nach erfolgreicher Erzeugung aller Kommunikationskanäle durch ein Modulobjekt sendet dieses eine Ausführungsmeldung an den veranlassenden Thread zurück. Sobald ein Thread von allen zugeordneten Modulobjekten die Ausführungsmeldung erhalten hat und auch selbst alle Kommunikationskanäle erfolgreich erzeugt hat, sendet der Thread ebenfalls eine Ausführungsmeldung an das übergeordneten Prozessobjekt. Hat ein Prozessobjekt von allen zugeordneten Threads eine Ausführungsmeldung empfangen, und auch selbst alle Kommunikationskanäle korrekt erzeugt, sendet es eine Ausführungsmeldung an das übergeordnete Applikationsobjekt. Sobald das Applikationsobjekt von allen zugeordneten Prozessobjekten eine Ausführungsmeldung erhalten hat, ist auch dieser zweite Schritt der Initialisierung abgeschlossen.

Da beim Betrieb der Steuerung zwischen den Objekten die unterschiedlichsten Daten möglichst schnell ausgetauscht werden müssen, werden die Kommunikationskanäle unterschiedlich realisiert. Beispielsweise kann ein Kommunikationskanal zwischen zwei Modulobjekten desselben Threads in Form eines für beide Modulobjekte gemeinsamen Speichers realisiert werden, über den Daten ausgetauscht werden; ein Kommunikationskanal zwischen Modulen in verschiedenen Prozessen muß Daten von einem Adressraum in einen anderen übertragen können.

Es besteht die Möglichkeit weitere Initialisierungsschritte durchzuführen, die prinzipiell identisch zu den bisher beschriebenen ablaufen. Es wird immer von den hierarchisch übergeordneten Objekten die Ausführung eines Programmteils zur Erzeugung einer bestimmten Funktion ausgelöst und anschliessend gewartet, bis von allen untergeordneten Objekten eine Ausführungsmeldung empfangen wurde, bevor eine Ausführungsmeldung an ein übergeordnetes Objekt gesendet wird. Erst nachdem von dem Applikationsobjekt die Ausführungsmeldungen aller zugeordneten Prozessobjekte empfangen wurden, erfolgt der nächste Schritt der Initialisierung. Dadurch wird sichergestellt, dass die Initialisierung synchron erfolgt und alle Objekt zu einem bestimmten Zeitpunkt einen definierten Zustand einnehmen.

In diesen weiteren Initialisierungsschritten können beispielsweise Parameter an einzelne Objekt übergeben werden. Die prinzipielle Vorgehensweise erfolgt wie bereits für den vorhergehenden Initialisierungsschritt beschrieben.

Als letzter Initialisierungsschritt erfolgt das Starten der Applikation. Dabei wird jedem Objekt signalisiert, dass die Initialisierung abgeschlossen wurde und von nun an der reguläre Betrieb der Steuerung beginnt. Dadurch erkennt ein Objekt, dass es die reguläre Bearbeitung von über die Kommunikationskanäle empfangenen Daten durchführen soll. Bei diesem letzten Schritt ist keine Ausführungsmeldung zum übergeordneten Objekt mehr zwingend erforderlich, da die korrekte Initialisierung bereits durch die vorhergehenden Ausführungsmeldungen signalisiert wurde und die zur Initialisierung erforderliche Synchronität nicht mehr benötigt wird.

Ein einfaches Beispiel für eine Initialisierung ist in **Fig. 2** dargestellt. Dabei wird zunächst das Ladeprogramm zum Laden des Startprogramms ausgeführt, so dass sich das Startprogramm im Arbeitsspeicher **4** befindet. Falls die Daten nicht bereits im Startprogramm enthalten sind, wird anschliessend durch das Startprogramm eine Datei geladen, die Informationen über die programmtechnische Struktur der gesamten Applikation und über eine oder mehrere Dateien beinhaltet, in der die gesamte Applikation auf dem Massenspeicher **1** abgelegt ist und die alle Daten für die gesamte zu initialisierende Applikation beinhaltet. Diese Information wird durch das Startprogramm derart benutzt, dass alle Dateien, die Informationen zur Applikation beinhalten, in den Arbeitsspeicher **4** geladen werden und das Applikationsobjekt **10** durch das Startprogramm erzeugt wird.

Dem derart erzeugten Applikationsobjekt **10** ist bekannt, dass die Prozesse **11, 12** und **13** dem Applikationsobjekt **10** zugeordnet sind. Daher werden durch das Applikationsobjekt **10** die Prozesse **11, 12** und **13** in der objektorientierten Struktur erzeugt. Dies geschieht dadurch, dass das Applikationsobjekt mindestens eine ausführbare Datei in den Arbeitsspeicher **4** lädt, und die Ausführung dieser mindestens einen Datei startet. Dadurch werden alle Prozessobjekte **11, 12** und **13** erzeugt.

Die neu erzeugten Prozessobjekte **11, 12** und **13** beinhalten Informationen darüber, welche Threads dem jeweiligen Prozessobjekt **11, 12** und **13** zugeordnet sind, so dass die Erzeugung der Threads **14, 15, 16** und **17** durch die Prozessobjekte **11–13** veranlasst wird.

Die Threadobjekte **14–17** beinhalten wiederum Informationen darüber, ob ihnen weitere Module zugeordnet sind. Dadurch werden die Modulobjekte **18–21** erzeugt. Die Modulobjekte **18–21** bilden die unterste Hierarchieebene der

Steuerungssoftware für die Werkzeugmaschine, so dass sie keine weitere Information über zugeordnete Objekte beinhalten.

In jedem Objekt ist gespeichert, welche weiteren Objekte ihm zugeordnet sind, beispielsweise ist im Applikationsobjekt **10** gespeichert, dass ihm die Prozessobjekte **11**, **12** und **13** zugeordnet sind, im Prozessobjekt **11** ist gespeichert, dass ihm die Threadobjekte **14** und **15** zugeordnet sind usw. Weiterhin ist in jedem Objekt gespeichert, von welchem Objekt es erzeugt worden ist, das heisst in den beiden Threadobjekten **14** und **15** ist gespeichert, dass sie von dem Prozessobjekt **11** erzeugt worden sind. Ausnahmen bilden das Applikationsobjekt **10**, in dem nicht gespeichert ist, von welchem Objekt es erzeugt worden ist, und die Modulobjekte **18–21**, denen keine weiteren Objekte zugeordnet sind.

Nachdem somit die gesamte Software der Steuerung synchronisiert erzeugt wurde, erfolgt noch eine Ausführungsmeldung durch die erzeugten Objekte an die Objekte, die sie erzeugt haben, über ihre erfolgreiche Erzeugung. Dies beginnt mit den zuletzt erzeugten Modulobjekten **18–21**, die nach einer erfolgreichen Erzeugung an die ihnen übergeordneten Threadobjekte **14–17**, von denen sie erzeugt wurden, eine Ausführungsmeldung übertragen. Diese Ausführungsmeldung signalisiert jedem der Threadobjekte **14–17** die erfolgreiche Erzeugung der von ihm erzeugten Modulobjekte **18–21** und wird von diesem nur übertragen wenn die Erzeugung der Modulobjekte **18–21** wirklich erfolgreich war.

Hat ein Threadobjekt die Ausführungsmeldung über die erfolgreiche Erzeugung aller ihm zugeordneten Modulobjekte empfangen und wurde auch selbst fehlerfrei erzeugt, sendet es ebenfalls eine Ausführungsmeldung, dass die Erzeugung erfolgreich abgeschlossen wurde an das Prozessobjekt, das den Threadobjekt erzeugt hat. Das bedeutet, dass die Threadobjekte **14** und **15** die Ausführungsmeldung an Prozessobjekt **11** senden, welches darauf bereits wartet. Sobald Prozessobjekt **11** von den von ihm erzeugten Threadobjekten **14** und **15** eine abgeschlossene Erzeugung gemeldete wurde, meldet auch Prozessobjekt **11** an das Applikationsobjekt **10** eine abgeschlossene Erzeugung. Ebenso meldet Threadobjekt **17** eine abgeschlossene Erzeugung an Prozessobjekt **13**, das danach wiederum eine abgeschlossene Erzeugung an das Applikationsobjekt **10** meldet.

Nachdem dem Applikationsobjekt **10** durch alle von ihm erzeugten Prozessobjekte **11**, **12** und **13** eine abgeschlossene Erzeugung der Struktur gemeldet wurde, steht die durch das Applikationsobjekt **10** realisierte Struktur in der Steuerung zur Verfügung. Der erste Schritt des synchronisierten Hochlaufs der Steuerung ist damit abgeschlossen.

Kann ein Objekt nicht erzeugt werden, so dass eine Störung vorliegt, sendet dieses Objekt keine Rückmeldung, dass die Erzeugung erfolgreich abgeschlossen wurde. Das hierarchisch übergeordnete Objekt stellt nach einer gewissen Wartezeit fest, dass der Hochlauf nicht erfolgreich durchgeführt werden konnte und sendet ebenfalls keine Ausführungsmeldung an das ihm übergeordnete Objekt. Die entsprechende Funktion steht dann in der Applikation nicht zur Verfügung. Aufgrund der in den jeweils übergeordneten Objekten nicht empfangenen Ausführungsmeldung kann ermittelt werden, an welcher Stelle eine Erzeugung nicht möglich war.

Es folgt in einem zweiten Schritt nun die Initialisierung der Kommunikationskanäle zwischen den Objekten, die nach der gleichen Systematik wie die Erzeugung der Struktur der Software abläuft. Es werden dabei für das Applikationsobjekt und jedes Prozess-, Thread- und Modulobjekt zumindest logische Kommunikationskanäle definiert, über die es mit bereits erzeugten Objekten kommunizieren kann. Dabei kann ein Kommunikationskanal auch wieder als ein Objekt angesehen werden. Zur Erzeugung der Kommunikationskanäle ist bereits in dem Applikationsobjekt, den Prozess-, Thread- und Modulobjekten gespeichert, mit welchen weiteren Objekten Nachrichten auszutauschen sind. Zu diesen Objekten werden dann Kommunikationskanäle erzeugt.

Nachdem die Struktur und die Kommunikationskanäle synchronisiert erzeugt wurden, erfolgt die synchronisierte Initialisierung der Applikation beispielsweise mit Startwerten oder einstellbaren Parametern. Dies erfolgt nach der gleichen Systematik, wie für den ersten Schritt bereits beschrieben.

Zum Abschluss des synchronisierten Hochlaufs wird allen Objekten der Übergang in den Normalbetrieb nach diesem Schema signalisiert. Auch dies erfolgt nach der gleichen, bereits oben beschriebenen Systematik.

Es ist für den Fachmann offensichtlich, dass die Aufteilung der Informationen auf eine oder mehrere Dateien beliebig erfolgen kann. Es besteht die Möglichkeit die benötigten Informationen in nur wenigen Dateien abzuspeichern. So kann beispielsweise das Startprogramm bereits weitgehende Informationen über die gesamte zu erzeugende Struktur beinhalten. Alternativ können auch viele kleinere Dateien vorgesehen werden, die jeweils nur einen kleinen Teil der benötigten Information für die Initialisierung beinhalten. Weiterhin können beim synchronisierten Hochlauf zusätzlich zur Erzeugung der Struktur, der Kommunikationskanäle, der Initialisierung und dem Übergang in den Normalbetrieb noch weitere Schritte durchlaufen werden, insbesondere abhängig von der durch die Applikation zu steuernden Werkzeugmaschine.

Insgesamt lässt sich der Hochlauf wie folgt zusammenfassen. Um eine einfache Wartbarkeit und Erweiterbarkeit sicherzustellen, weisen die Programme für numerische Steuerungen zunehmend eine objektorientierte Struktur auf. Dabei werden Applikationen, Prozesse, Threads und Module jeweils durch ein Objekt repräsentiert. Bei einem Hochlauf werden zunächst diejenigen Objekte erzeugt, die weitere Objekte erzeugen, und es nimmt jedes Objekt eine Anzahl definierter Zustände an. Insbesondere startet jedes Objekt bei allen von ihm erzeugten Objekten jede durch das jeweilige Objekt realisierte Initialisierungsfunktion und nimmt den nächsten definierten Zustand dann an, wenn es selbst und alle von ihm erzeugten Objekte sämtliche Funktionen vollständig ausgeführt haben. Dieser Zustand wird an das erzeugende Objekt gemeldet, bis diese Rückmeldung beim Applikationsobjekt eintrifft. Das Applikationsobjekt nimmt dann selbst diesen Zustand an und löst den nächsten Schritt des Hochlaufs aus bzw. beendet den Hochlauf, nachdem der letzte Schritt des Hochlaufs ausgeführt wurde.

Patentansprüche

1. Verfahren zum synchronisierten Hochlauf einer Applikation, insbesondere für eine numerische Steuerung einer Werkzeugmaschine oder eines Roboters, bei dem einzelne Initialisierungsschritte, bei denen gleichartige Massnahmen zur Initialisierung der Applikation durchgeführt werden, in der gesamten Applikation im wesentlichen zeitlich synchron ausgeführt werden.
2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass zunächst ein Startprogramm ausgeführt wird, das Informationen über die Applikation und über die zeitliche Reihenfolge der einzelnen Initialisierungsschritte beinhaltet und das zumindest einen Teil der Daten der Applikation von einem Massenspeicher **(1)** in einen Arbeitsspeicher **(4)** lädt.
3. Verfahren nach einem der Ansprüche 1 oder 2, dadurch gekennzeichnet, dass in einem Initialisierungs-

schritt in allen Objekten gleichartige aber nicht zwingend identische Massnahmen zur Initialisierung durchgeführt werden und dass in aufeinanderfolgenden Initialisierungsschritten ungleichartige Massnahmen zur Initialisierung durchgeführt werden.

4. Verfahren nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, dass die in einem Initialisierungsschritt durchzuführenden Massnahmen in dem Programmteil oder Objekt gespeichert sind, auf das sie auch angewendet werden.

5. Verfahren nach einem der Ansprüche 1 bis 4, dadurch gekennzeichnet, dass die Ausführung eines Initialisierungsschritts in einem Programmteil oder in einem Objekt bei einer Applikation von einem Startprogramm und sonst von dem jeweils hierarchisch übergeordneten Objekt veranlasst wird.

6. Verfahren nach einem der Ansprüche 1 bis 5, dadurch gekennzeichnet, dass nach der korrekten Durchführung eines Initialisierungsschritts in einem Objekt und nach dem Empfang der Ausführungsmeldungen von allen dem jeweiligen Objekt zugeordneten Objekten, dieses Objekt an das hierarchisch übergeordnete Objekt eine Ausführungsmeldung weiterleitet.

7. Verfahren nach einem der Ansprüche 1 bis 6, dadurch gekennzeichnet, dass das Startprogramm oder die Applikation den nächsten Initialisierungsschritt erst startet, nachdem es von allen Applikationsobjekten oder Prozessobjekten eine Ausführungsmeldung empfangen hat.

8. Verfahren nach einem der Ansprüche 1 bis 7, dadurch gekennzeichnet, dass in einem Initialisierungsschritt einzelne, im wesentlichen abgeschlossene Funktionen der Applikation in Form von Objekten erzeugt werden.

9. Verfahren nach einem der Ansprüche 1 bis 8, dadurch gekennzeichnet, dass in einem Initialisierungsschritt Kommunikationskanäle zwischen Objekten der Applikation erzeugt werden.

10. Verfahren nach einem der Ansprüche 1 bis 9, dadurch gekennzeichnet, dass in einem Initialisierungsschritt den Variablen in den einzelnen Objekten Parameter zugewiesen werden.

11. Verfahren nach einem der Ansprüche 1 bis 10, dadurch gekennzeichnet, dass in einem Initialisierungsschritt der reguläre Betrieb der Applikation freigegeben wird.

Hierzu 1 Seite(n) Zeichnungen

- Leerseite -

Nummer: DE 199 24 461 A1
Int. Cl.[7]: G 05 B 19/414
Offenlegungstag: 30. November 2000

FIG. 1

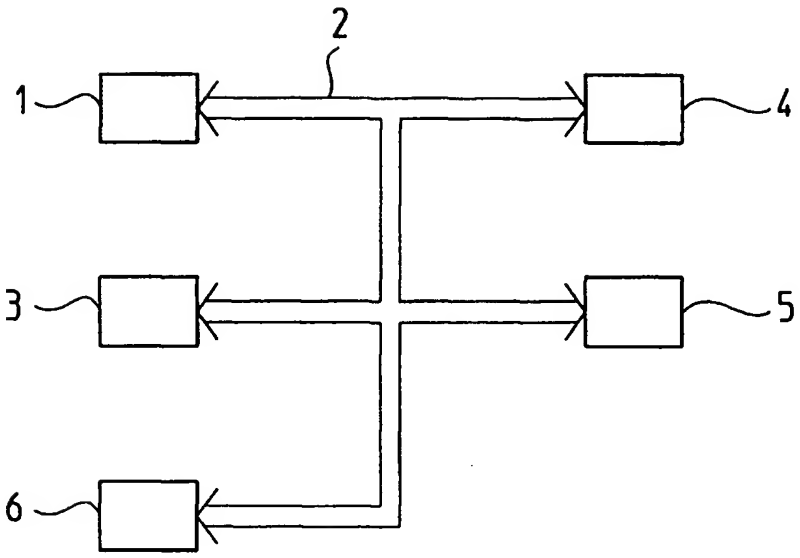


FIG. 2